CS 6375: Machine Learning - Project 3

Deep Learning for MNIST and CIFAR-10

Instructor : Tahrima Rahman

Navaneeta Padmakumar / NXP230016

11/08/2025

1. Introduction

This project implements and evaluates Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) on two benchmark image classification datasets: MNIST and CIFAR-10. We systematically explore various architectures and hyperparameters to understand the impact of network depth, optimization strategies, and regularization techniques on model performance.

2. Datasets and Preprocessing

2.1 MNIST Dataset

- **Training set:** 50,000 samples
- **Validation set:** 10,000 samples
- **Test set:** 10,000 samples
- **Image size:** 28×28 grayscale images
- **Number of classes:** 10 (digits 0-9)

2.2 CIFAR-10 Dataset

- **Training set:** 45,000 samples
- **Validation set:** 5,000 samples
- **Test set:** 10,000 samples
- **Image size:** 32×32×3 RGB images
- **Number of classes:** 10 (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)

2.3 Preprocessing Steps

- Pixel values normalized to [0,1] range by dividing by 255
- Images flattened to 1D vectors for MLP input (784 for MNIST, 3072 for CIFAR-10)
- Images kept in 2D format for CNN input
- Used PyTorch's torchvision.datasets for data loading
- Applied torch.utils.data.random_split with manual seed (42) for reproducible train/validation splits

3. Experimental Setup

3.1 Common Configuration

- Framework: PyTorch
- Device: CUDA GPU (Google Colab T4 GPU)
- Activation function: ReLU for all hidden layers
- Loss function: CrossEntropyLoss
- Number of epochs: 10 (with early stopping when applicable)
- Early stopping patience: 3 epochs

3.2 Hyperparameter Search Space

For each architecture, we explored 8 meaningful configurations varying:

- Learning rate: {0.0001, 0.001, 0.01}
- Batch size: {64, 128}
- Optimizer: {Adam, SGD}
- Dropout rate: {0.0, 0.2, 0.3, 0.5}

4. Part 1: Multilayer Perceptrons (MLPs)

4.1 MLP Architectures

We implemented three MLP architectures of varying depth:

1. **Shallow MLP (1 hidden layer):**
   Architecture: Input → Dense(256) → ReLU → Dropout → Dense(10)
   Parameters: ~200K for MNIST, ~800K for CIFAR-10

2. **Medium MLP (3 hidden layers):**
   Architecture: Input → Dense(512) → ReLU → Dropout → Dense(256) → ReLU → Dropout → Dense(128) → ReLU → Dropout → Dense(10)
   Parameters: ~470K for MNIST, ~1.6M for CIFAR-10

3. **Deep MLP (5 hidden layers):**
   Architecture: Input → Dense(512) → ReLU → Dropout → Dense(256) → ReLU → Dropout → Dense(256) → ReLU → Dropout → Dense(128) → ReLU → Dropout → Dense(64) → ReLU → Dropout → Dense(10)
   Parameters: ~830K for MNIST, ~2.2M for CIFAR-10

4.2 MNIST MLP Results

Table 1a: MNIST - Shallow MLP (1 Hidden Layer)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.2	97.31	1.15
2	0.01	128	Adam	0.2	96.60	1.05
3	0.0001	128	Adam	0.2	93.06	1.18
4	0.001	64	Adam	0.2	97.58	1.40
5	0.001	128	SGD	0.2	95.12	1.12
6	0.001	128	Adam	0.0	97.21	1.10
7	0.001	128	Adam	0.5	96.89	1.14
8	0.01	64	Adam	0.2	96.45	1.08

Best Configuration: Config 4 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 96.28 ± 1.42%

Table 1b: MNIST - Medium MLP (3 Hidden Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.2	97.69	1.08
2	0.01	128	Adam	0.2	96.63	1.21
3	0.0001	128	Adam	0.2	96.61	1.21
4	0.001	64	Adam	0.2	97.85	1.47
5	0.001	128	SGD	0.2	95.45	1.15
6	0.001	128	Adam	0.0	97.51	1.12
7	0.001	128	Adam	0.5	96.98	1.19
8	0.01	64	Adam	0.2	96.72	1.11

Best Configuration: Config 4 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 96.93 ± 0.73%

Table 1c: MNIST - Deep MLP (5 Hidden Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.2	97.72	1.25
2	0.01	128	Adam	0.2	94.69	0.74
3	0.0001	128	Adam	0.2	96.32	1.25
4	0.001	64	Adam	0.2	97.98	1.55
5	0.001	128	SGD	0.2	11.88	0.68
6	0.001	128	Adam	0.0	97.65	1.22
7	0.001	128	Adam	0.5	97.12	1.28
8	0.01	64	Adam	0.2	94.85	0.76

Best Configuration: Config 4 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 86.03 ± 28.97% (Note: High std due to SGD failure in Config 5)

Table 1d: MNIST MLP Summary

Architecture	Learning Rate	Batch	Optimizer	Dropout	Val Acc (Mean±Std)	Runtime (min)
Shallow MLP (1 hidden)	0.001	64	Adam	0.2	96.28 ± 1.42%	1.40
Medium MLP (3 hidden)	0.001	64	Adam	0.2	96.93 ± 0.73%	1.47
Deep MLP (5+ hidden)	0.001	64	Adam	0.2	86.03 ± 28.97%	1.55

Test Accuracy on Final MLP Model (Deep MLP): 98.09%

4.3 CIFAR-10 MLP Results

Table 2a: CIFAR-10 - Shallow MLP (1 Hidden Layer)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.2	44.46	1.19
2	0.01	128	Adam	0.2	42.13	1.15
3	0.0001	128	Adam	0.2	41.89	1.22
4	0.001	64	Adam	0.2	43.98	1.35
5	0.001	128	SGD	0.2	38.76	1.18
6	0.001	128	Adam	0.0	43.12	1.16
7	0.001	128	Adam	0.5	42.56	1.20
8	0.01	64	Adam	0.2	41.34	1.12

Best Configuration: Config 1 (LR=0.001, Batch=128, Adam, Dropout=0.2)

Mean ± Std: 42.28 ± 1.65%

Table 2b: CIFAR-10 - Medium MLP (3 Hidden Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.0	47.92	1.20
2	0.01	128	Adam	0.0	45.67	1.18
3	0.0001	128	Adam	0.0	44.23	1.24
4	0.001	64	Adam	0.0	47.45	1.38
5	0.001	128	SGD	0.0	41.12	1.19
6	0.001	128	Adam	0.2	46.87	1.22
7	0.001	128	Adam	0.5	45.34	1.23
8	0.01	64	Adam	0.0	44.98	1.15

Best Configuration: Config 1 (LR=0.001, Batch=128, Adam, Dropout=0.0)

Mean ± Std: 45.45 ± 2.08%

Table 2c: CIFAR-10 - Deep MLP (5 Hidden Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.0	48.26	1.23
2	0.01	128	Adam	0.0	46.12	1.19
3	0.0001	128	Adam	0.0	45.34	1.27
4	0.001	64	Adam	0.0	47.89	1.42
5	0.001	128	SGD	0.0	10.23	0.85
6	0.001	128	Adam	0.2	47.45	1.25
7	0.001	128	Adam	0.5	45.98	1.26
8	0.01	64	Adam	0.0	45.67	1.17

Best Configuration: Config 1 (LR=0.001, Batch=128, Adam, Dropout=0.0)

Mean ± Std: 42.12 ± 13.98% (Note: High std due to SGD failure in Config 5)

Table 2d: CIFAR-10 MLP Summary

Architecture	Learning Rate	Batch	Optimizer	Dropout	Val Acc (Mean±Std)	Runtime (min)
Shallow MLP (1 hidden)	0.001	128	Adam	0.2	42.28 ± 1.65%	1.19
Medium MLP (3 hidden)	0.001	128	Adam	0.0	45.45 ± 2.08%	1.20
Deep MLP (5+ hidden)	0.001	128	Adam	0.0	42.12 ± 13.98%	1.23

Test Accuracy on Final MLP Model (Deep MLP): 49.57%

5. Part 2: Convolutional Neural Networks (CNNs)

5.1 CNN Architectures

We implemented three CNN architectures of varying complexity:

1. **Baseline CNN (2 convolutional layers):**
   - Conv2d(in, 32, 3) → ReLU → MaxPool2d(2)
   - Conv2d(32, 64, 3) → ReLU → MaxPool2d(2)
   - Flatten → Dense(128) → ReLU → Dropout → Dense(10)

   Parameters: ~150K for MNIST

2. **Enhanced CNN (with Batch Normalization):**
   - Conv2d(in, 32, 3) → BatchNorm2d(32) → ReLU → MaxPool2d(2) → Dropout2d(0.1) → Conv2d(32, 64, 3) → BatchNorm2d(64) → ReLU → MaxPool2d(2) → Dropout2d(0.1) → Flatten → Dense(128) → BatchNorm1d(128) → ReLU → Dropout(0.2) → Dense(10)
   - Conv2d(in, 32, 3) → BatchNorm2d(32) → ReLU → MaxPool2d(2) → Dropout2d(0.1) → Conv2d(32, 64, 3) → BatchNorm2d(64) → ReLU → MaxPool2d(2) → Dropout2d(0.1) → Flatten → Dense(128) → BatchNorm1d(128) → ReLU → Dropout(0.2) → Dense(10)
   - Flatten → Dense → BatchNorm1d → ReLU → Dropout → Dense(10)

   Parameters: ~180K for MNIST

3. **Deeper CNN (5 convolutional layers):**
   - 2× Conv2d(32) → Pool → Dropout2d(0.1)
   - 2× Conv2d(64) → Pool → Dropout2d(0.2)
   - 1× Conv2d(128) → Pool → Dropout2d(0.2)
   - 2× FC layers

   Parameters: ~1.2M for MNIST

5.2 MNIST CNN Results

Table 3a: MNIST - Baseline CNN (2 Conv Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.01	128	Adam	0.2	96.43	1.31
2	0.001	128	Adam	0.2	98.67	1.35
3	0.0001	128	Adam	0.2	97.23	1.38
4	0.01	64	Adam	0.2	96.89	1.52
5	0.001	128	SGD	0.2	97.12	1.32
6	0.001	128	Adam	0.0	98.45	1.30
7	0.001	128	Adam	0.5	98.21	1.34
8	0.001	64	Adam	0.2	98.78	1.48

Best Configuration: Config 8 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 97.72 ± 0.91%

Table 3b: MNIST - Enhanced CNN (BN+Dropout)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.01	128	Adam	0.2	94.76	1.05
2	0.001	128	Adam	0.2	98.45	1.08
3	0.0001	128	Adam	0.2	97.56	1.12
4	0.01	64	Adam	0.2	95.23	1.18
5	0.001	128	SGD	0.2	97.34	1.06
6	0.001	128	Adam	0.0	98.32	1.04
7	0.001	128	Adam	0.5	98.12	1.09
8	0.001	64	Adam	0.2	98.67	1.21

Best Configuration: Config 8 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 97.31 ± 1.45%

Table 3c: MNIST - Deeper CNN (5 Conv Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	128	Adam	0.2	99.16	1.47
2	0.01	128	Adam	0.2	98.45	1.42
3	0.0001	128	Adam	0.2	98.67	1.52
4	0.001	64	Adam	0.2	99.23	1.68
5	0.001	128	SGD	0.2	98.12	1.45
6	0.001	128	Adam	0.0	98.98	1.43
7	0.001	128	Adam	0.5	98.76	1.49
8	0.01	64	Adam	0.2	98.34	1.40

Best Configuration: Config 4 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 98.71 ± 0.40%

Table 3d: MNIST CNN Summary

Architecture	Learning Rate	Batch	Optimizer	Dropout	Val Acc (Mean±Std)	Runtime (min)
Baseline CNN (2 conv)	0.001	64	Adam	0.2	97.72 ± 0.91%	1.48
Enhanced CNN (BN+dropout)	0.001	64	Adam	0.2	97.31 ± 1.45%	1.21
Deeper CNN (≥3 conv)	0.001	64	Adam	0.2	98.71 ± 0.40%	1.68

Test Accuracy on Final CNN Model (Deeper CNN): 99.25%

5.3 CIFAR-10 CNN Results

Table 4a: CIFAR-10 - Baseline CNN (2 Conv Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.01	128	Adam	0.2	45.38	1.29
2	0.001	128	Adam	0.2	58.67	1.35
3	0.0001	128	Adam	0.2	54.23	1.42
4	0.01	64	Adam	0.2	46.12	1.48
5	0.001	128	SGD	0.2	52.34	1.31
6	0.001	128	Adam	0.0	57.89	1.28
7	0.001	128	Adam	0.5	56.45	1.33
8	0.001	64	Adam	0.2	59.12	1.52

Best Configuration: Config 8 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 53.78 ± 5.25%

Table 4b: CIFAR-10 - Enhanced CNN (BN+Dropout)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	64	Adam	0.2	49.80	1.34
2	0.01	64	Adam	0.2	46.23	1.28
3	0.0001	64	Adam	0.2	47.56	1.38
4	0.001	128	Adam	0.2	48.92	1.25
5	0.001	64	SGD	0.2	44.67	1.32
6	0.001	64	Adam	0.0	49.12	1.31
7	0.001	64	Adam	0.5	47.89	1.36
8	0.01	128	Adam	0.2	45.98	1.22

Best Configuration: Config 1 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 47.52 ± 1.65%

Table 4c: CIFAR-10 - Deeper CNN (5 Conv Layers)

Config	LR	Batch	Optimizer	Dropout	Val Acc (%)	Runtime (min)
1	0.001	64	Adam	0.2	72.00	1.61
2	0.01	64	Adam	0.2	68.45	1.55
3	0.0001	64	Adam	0.2	69.23	1.68
4	0.001	128	Adam	0.2	71.34	1.52
5	0.001	64	SGD	0.2	65.12	1.58
6	0.001	64	Adam	0.0	70.89	1.57
7	0.001	64	Adam	0.5	69.67	1.63
8	0.01	128	Adam	0.2	67.89	1.48

Best Configuration: Config 1 (LR=0.001, Batch=64, Adam, Dropout=0.2)

Mean ± Std: 69.32 ± 2.25%

Table 4d: CIFAR-10 CNN Summary

Architecture	Learning Rate	Batch	Optimizer	Dropout	Val Acc (Mean±Std)	Runtime (min)
Baseline CNN (2 conv)	0.001	64	Adam	0.2	53.78 ± 5.25%	1.52
Enhanced CNN (BN+dropout)	0.001	64	Adam	0.2	47.52 ± 1.65%	1.34
Deeper CNN (≥3 conv)	0.001	64	Adam	0.2	69.32 ± 2.25%	1.61

Test Accuracy on Final CNN Model (Deeper CNN): 73.24%

6. Discussion

6.1 Why Certain Hyperparameters Performed Better

Learning Rate:
The learning rate of 0.001 emerged as optimal across most configurations. LR=0.0001 was too conservative, leading to slow convergence. LR=0.01 was too aggressive for deeper networks, causing instability and potential divergence.

Optimizer Choice:
Adam consistently outperformed SGD because of its adaptive learning rates for each parameter, built-in momentum, and forgiveness to learning rate selection. SGD requires careful tuning and often benefits from learning rate schedules.

Dropout Regularization:
Dropout effectiveness varied by task. For MNIST, moderate dropout (0.2) helped prevent overfitting. For CIFAR-10, no dropout or minimal dropout performed best for MLPs, as the task is inherently more challenging and models need full capacity. High dropout (0.5) consistently hurt performance.

6.2 CNNs vs MLPs Performance Comparison

MNIST Results:
CNNs achieved 99.25% on MNIST compared to MLPs' 98.09%, representing a 1.16% absolute improvement. This modest improvement demonstrates that CNNs effectively capture local spatial patterns crucial for digit recognition, with parameter efficiency through weight sharing and built-in translation invariance.

CIFAR-10 Results:
CNNs achieved 73.24% on CIFAR-10 compared to MLPs' 49.57%, representing a dramatic 23.67% absolute improvement. This massive gap demonstrates that spatial inductive biases in CNNs are essential for complex visual tasks. MLPs on CIFAR-10 barely exceeded random guessing, while CNNs achieved respectable performance.

6.3 Key Challenge and Resolution

Challenge:

One significant challenge was the extremely poor performance of Deep MLP with SGD optimizer on MNIST (11.88% accuracy) and similar failures on deeper networks.

Root Cause:

SGD with default momentum on deep networks without batch normalization led to gradient flow issues. The combination of deep architecture (5 layers) and basic SGD caused vanishing/exploding gradients.

Resolution:

- Switched to Adam optimizer for deep architectures
- For configurations requiring SGD, reduced learning rate and added gradient clipping
- Implemented early stopping to avoid wasting computation on clearly diverging configurations
- Added batch normalization to Enhanced CNN architecture to stabilize training

This experience highlighted the importance of optimizer selection for deep architectures, the value of adaptive optimization methods like Adam for complex models and reinforced that proper hyperparameter tuning is often more critical than architecture complexity alone.

7. Conclusion

This project successfully demonstrated several key findings in deep learning:

- MLPs can achieve strong performance on simple datasets like MNIST (98.09%) but struggle dramatically with spatially complex tasks like CIFAR-10 (49.57%)
- CNNs dramatically outperform MLPs by leveraging spatial structure, with improvements scaling with task complexity: +1.16% on MNIST, +23.67% on CIFAR-10
- The performance gap between MLPs and CNNs scales with task complexity: minimal on simple MNIST, dramatic on complex CIFAR-10
- Deeper architectures generally improve performance, but require careful hyperparameter tuning and appropriate optimization strategies
- Adam optimizer is more robust than SGD for deep learning tasks, especially without batch normalization
- Moderate learning rates (0.001) and appropriate regularization are crucial for optimal performance

7.1 Final Results Summary

Table 5: Final Test Accuracies

Dataset	Best MLP	Best CNN	Improvement
MNIST	98.09%	99.25%	+1.16%
CIFAR-10	49.57%	73.24%	+23.67%

The systematic hyperparameter exploration revealed that success in deep learning depends on the careful interplay between architecture design, optimization strategy, and regularization techniques. Most importantly, this project demonstrates that architectural choices must match the inherent structure of the data—CNNs are not merely better than MLPs for images, they are fundamentally necessary for complex visual recognition tasks.